2. Шаг за шагом

Узлы и Сцены

Мы увидели, что игра на Godot представляет собой дерево сцен, и что каждая сцена представляет собой дерево узлов. В этом уроке мы расскажем о них немного побольше. Также вы создадите свою первую сцену.

Узлы

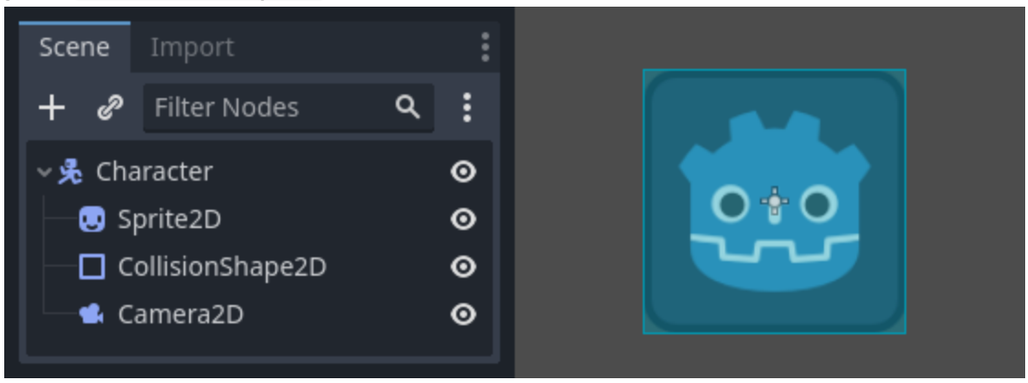
Узлы — это фундаментальные строительные блоки вашей игры. Они подобны ингредиентам в рецепте. Существуют десятки их видов, которые могут отображать изображение, воспроизводить звук, представлять камеру и многое другое.

Все узлы обладают следующими характеристиками:

- Имя.
- Редактируемые свойства.
- Они получают обратные вызовы для обновления каждого кадра.
- Вы можете расширить их новыми свойствами и функциями.
- Вы можете добавить их в другой узел в качестве дочерних.

Последняя характеристика очень важна. **Вместе узлы образуют дерево**, которое является мощной функцией для организации проектов. Поскольку разные узлы имеют разные функции, их объединение позволяет получить более сложное поведение. Как мы уже видели, вы можете создать играбельного персонажа, за которым следует камера, используя узел `CharacterBody2D`, узел `Sprite2D`, узел `Camera2D` и

узел `CollisionShape2D`.



Сцены

Когда вы организуете узлы в дереве, подобно нашему персонажу, мы называем эту конструкцию сценой. После сохранения сцены работают как новые типы узлов в редакторе, где вы можете добавить их в качестве дочернего элемента существующего узла. В этом случае экземпляр сцены отображается как отдельный узел со скрытыми внутренними элементами.

Сцены позволяют вам структурировать код вашей игры так, как вы этого хотите. Вы можете **компоновать узлы** для создания произвольных и сложных типов узлов, таких как игровой персонаж, который бегает и прыгает, полоса жизни, сундук, с которым можно взаимодействовать, и многое другое.

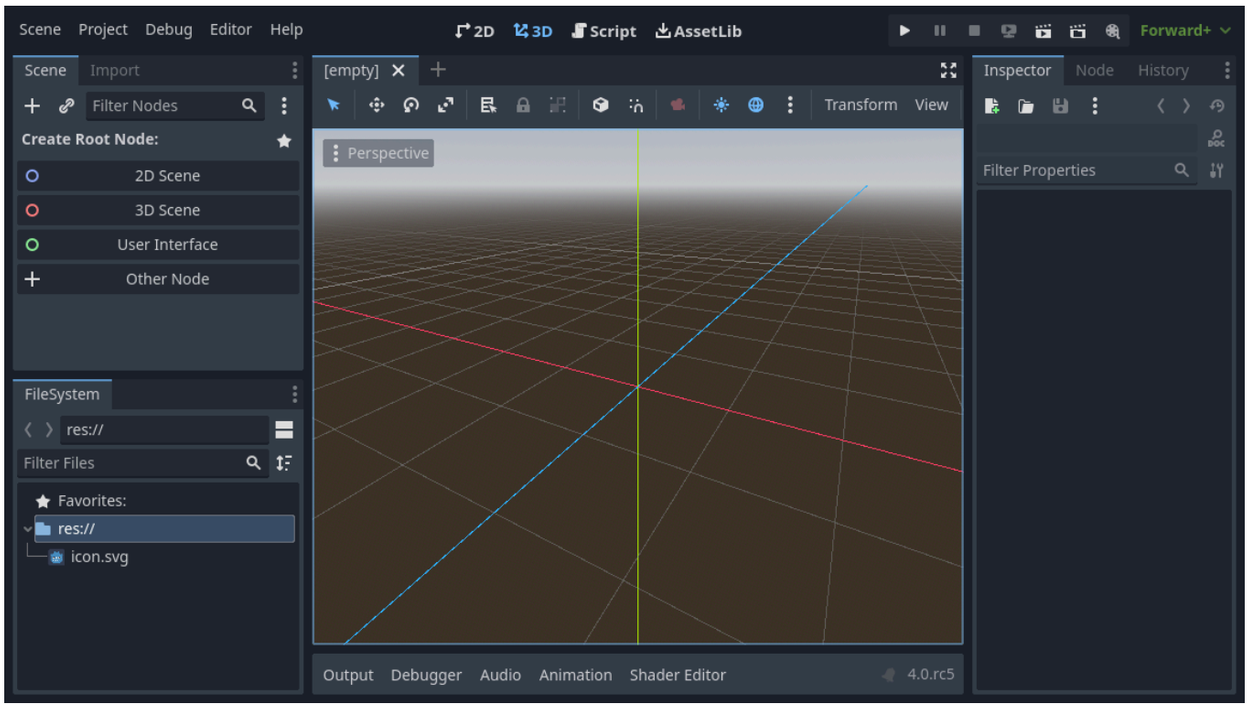
По сути, редактор Godot - это **редактор сцен**. В нем есть множество инструментов для редактирования 2D- и 3D-сцен, а также пользовательских интерфейсов. Проект Godot может содержать столько сцен, сколько вам необходимо. Движку же требуется только одна сцена - **главная сцена**. Это сцена, которую Godot загрузит первой, когда вы или игрок запустите игру.

Помимо того, что сцены действуют как узлы, они обладают следующими характеристиками:

1. Они всегда имеют один корневой узел, такой как "Player" в нашем примере.
2. Вы можете сохранить их на вашем локальном диске и загрузить позже.
3. Вы можете создать столько экземпляров сцены, сколько захотите. В вашей игре может быть пять или десять персонажей, созданных на основе сцены "Персонаж".

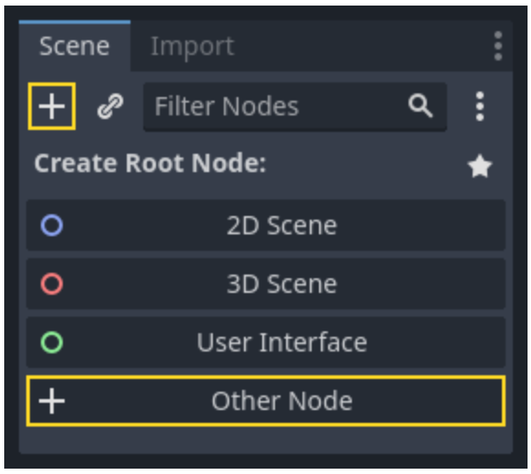
Создание вашей первой сцены

Давайте создадим нашу первую сцену с одним узлом. Для этого сначала нужно создать новый проект. После открытия проекта вы должны увидеть пустой редактор.

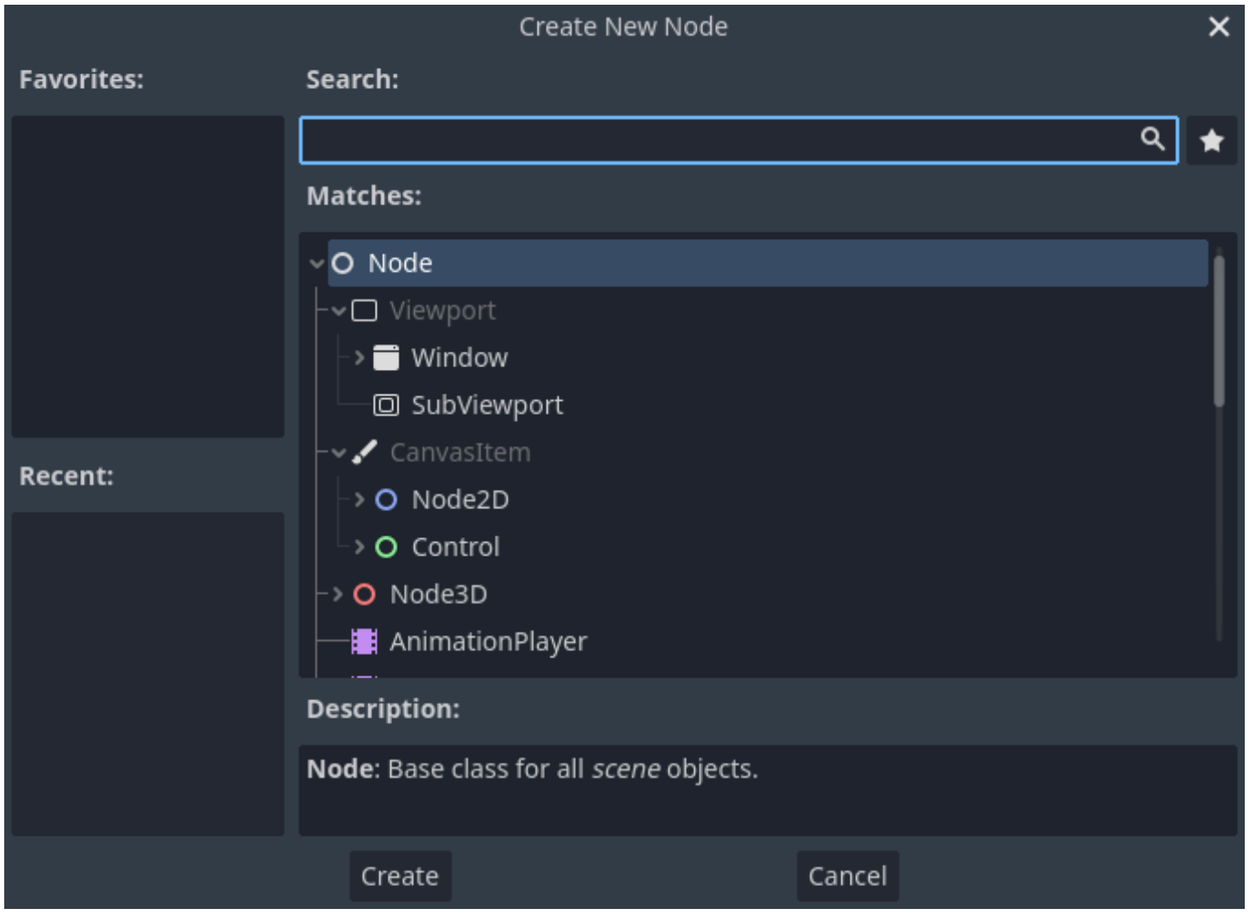


В пустой сцене в доке Scene слева показано несколько вариантов быстрого добавления корневого узла. "2D Scene" добавляет узел Node2D, "3D Scene" добавляет узел Node3D, а "User Interface" добавляет узел Control. Эти предустановки приведены для удобства; они не являются обязательными. "Other Node" позволяет выбрать любой узел в качестве корневого. В пустой сцене "Other Node" эквивалентен нажатию кнопки "Add Child Node" (Добавить дочерний узел) в левом верхнем углу дока Scene, которая обычно добавляет новый узел в качестве дочернего узла текущего выбранного узла.

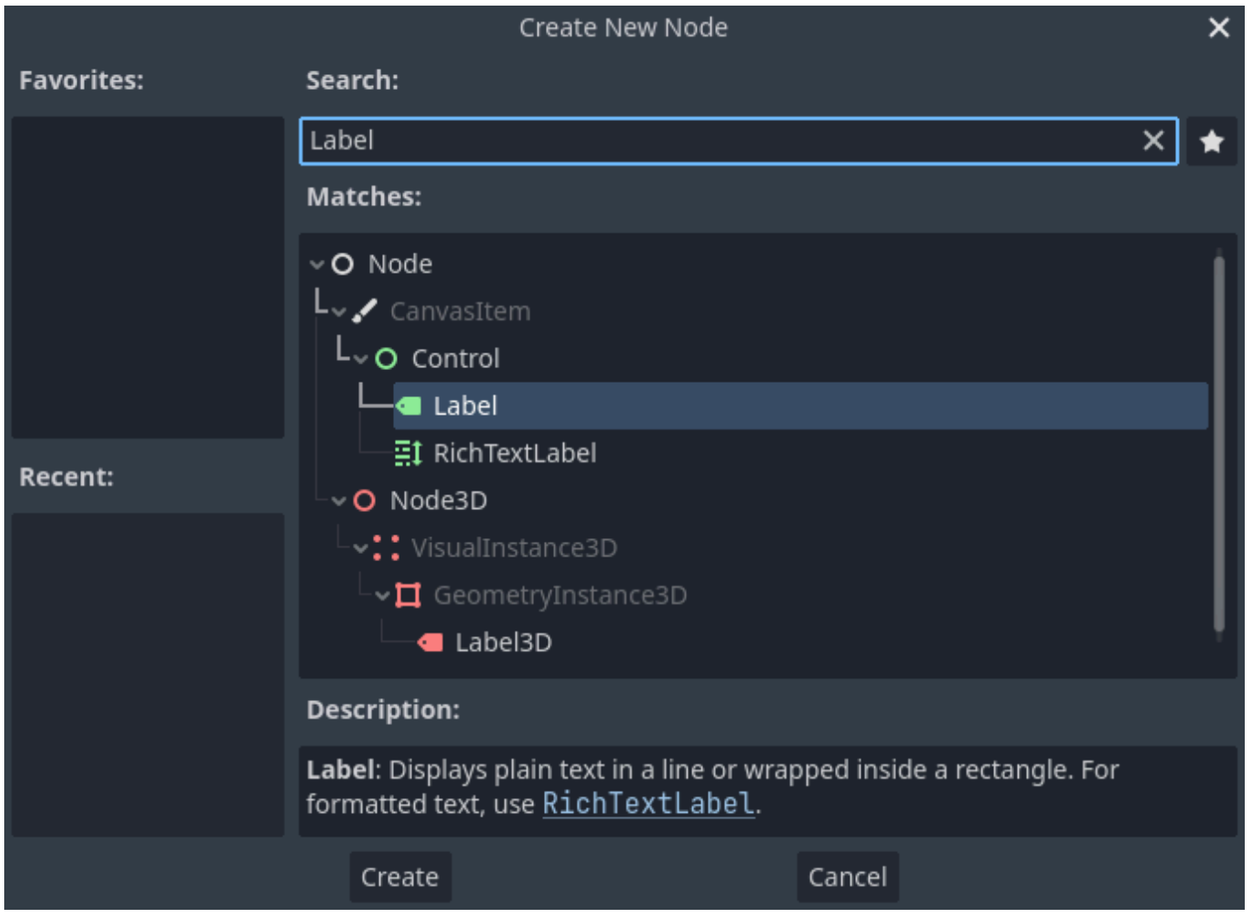
Мы добавим в нашу сцену один узел Label. Его функция - рисовать текст на экране. Нажмите кнопку "Add Child Node" (+) или "Other Node", чтобы создать корневой узел.



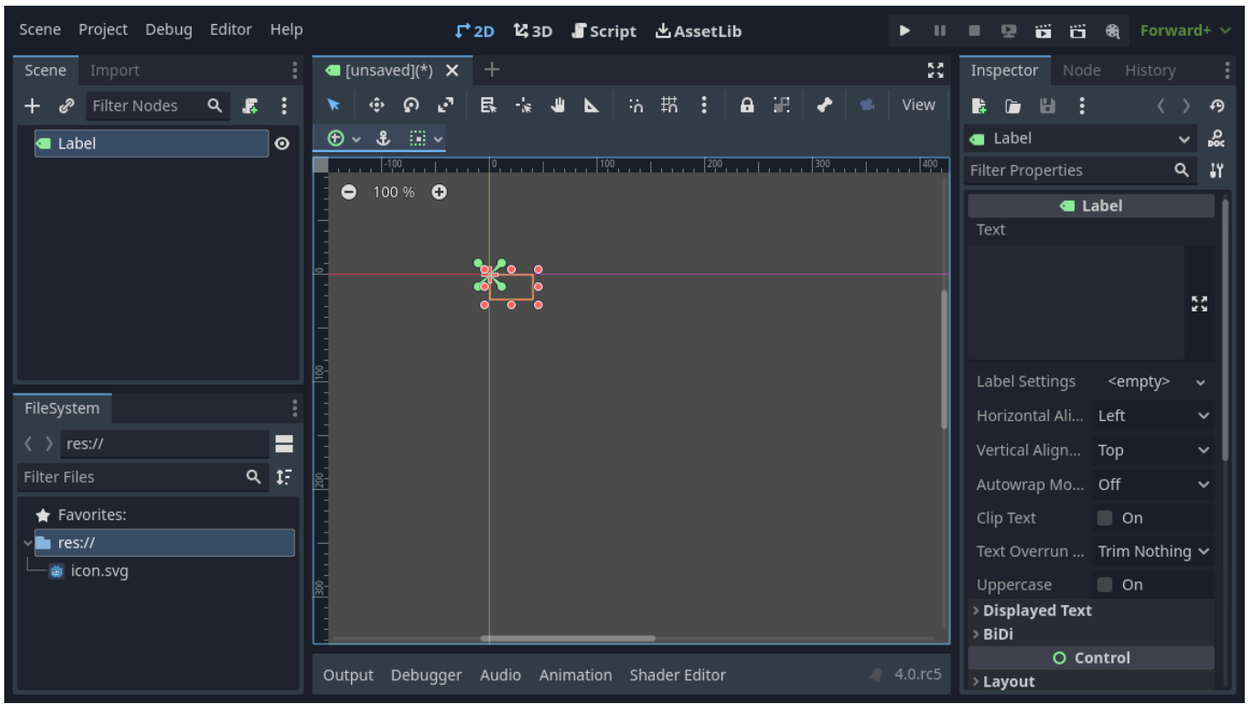
Откроется диалоговое окно "Create Node" (Создать узел), в котором отобразится длинный список доступных узлов.



Выберите узел Label (Метка). Вы можете ввести его имя, чтобы отфильтровать его по списку.



Щелкните на узле Label, чтобы выделить его, и нажмите кнопку Create в нижней части окна.

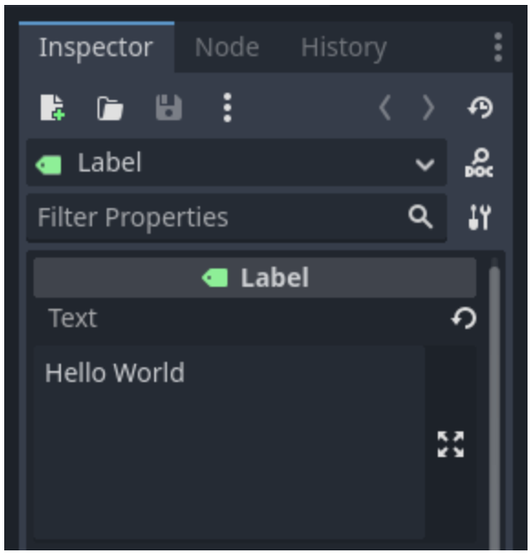


Многое происходит, когда вы добавляете первый узел сцены. Сцена изменится на 2D-пространство, поскольку Label является 2D-узлом. Label обозначается выбранной в верхнем левом углу окна просмотра. Узел появится во вкладке Scene слева, а свойства узла — во вкладке Inspector справа.

Изменение свойств узла

Следующим шагом будет изменение свойства Label's "Text". Давайте изменим его на "Hello World".

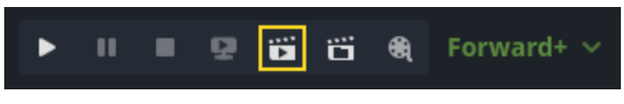
Перейдите на вкладку Inspector (справа от окна просмотра). Щелкните внутри поля под свойством Text и введите "Hello World".



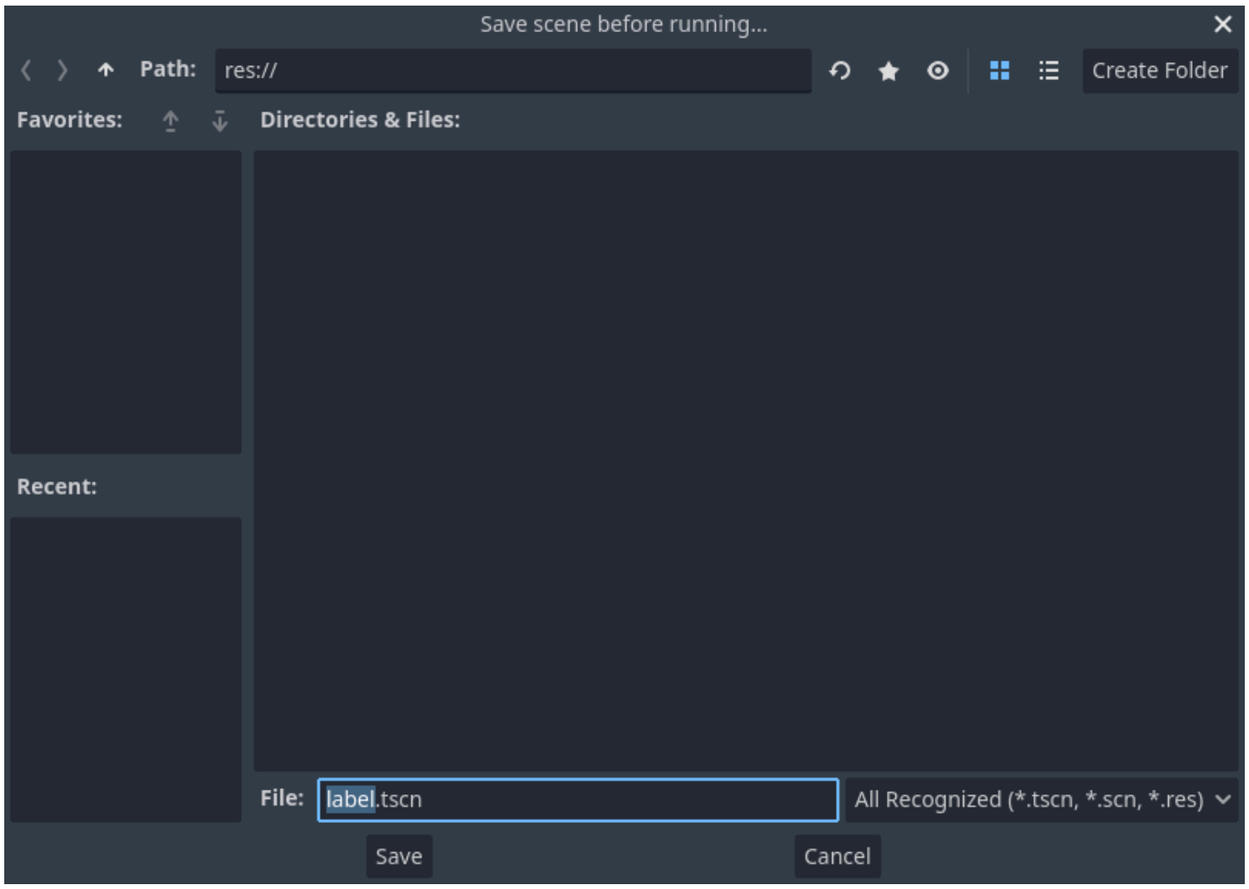
Вы увидите, как текст отрисовывается в окне просмотра по мере ввода.
Узел Label можно переместить в окне просмотра, выбрав инструмент перемещения на панели инструментов.
Когда метка выделена, нажмите и перетащите в любом месте области просмотра, чтобы переместить ее в центр ограниченного прямоугольником вида.

Запуск сцены

Все готово для запуска сцены! Нажмите кнопку Play Scene в правом верхнем углу экрана или нажмите F6



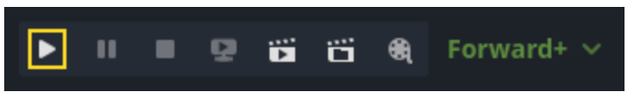
Во всплывающем окне предлагается сохранить сцену, что необходимо для её запуска. Нажмите кнопку сохранения в обозревателе файлов, чтобы сохранить её как `label.tscn`.



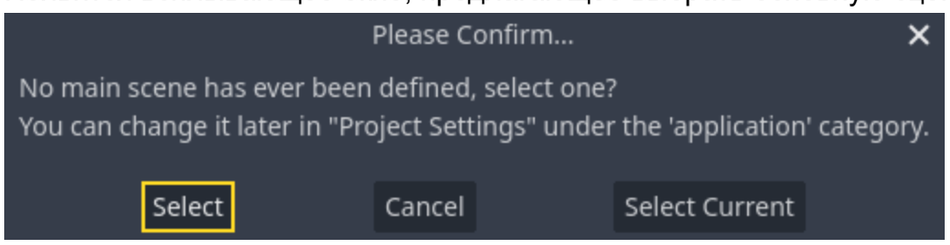
Приложение должно открыться в новом окне и отобразить текст «Hello World». Закройте окно чтобы выйти из запущенной сцены.

Настройка главной сцены

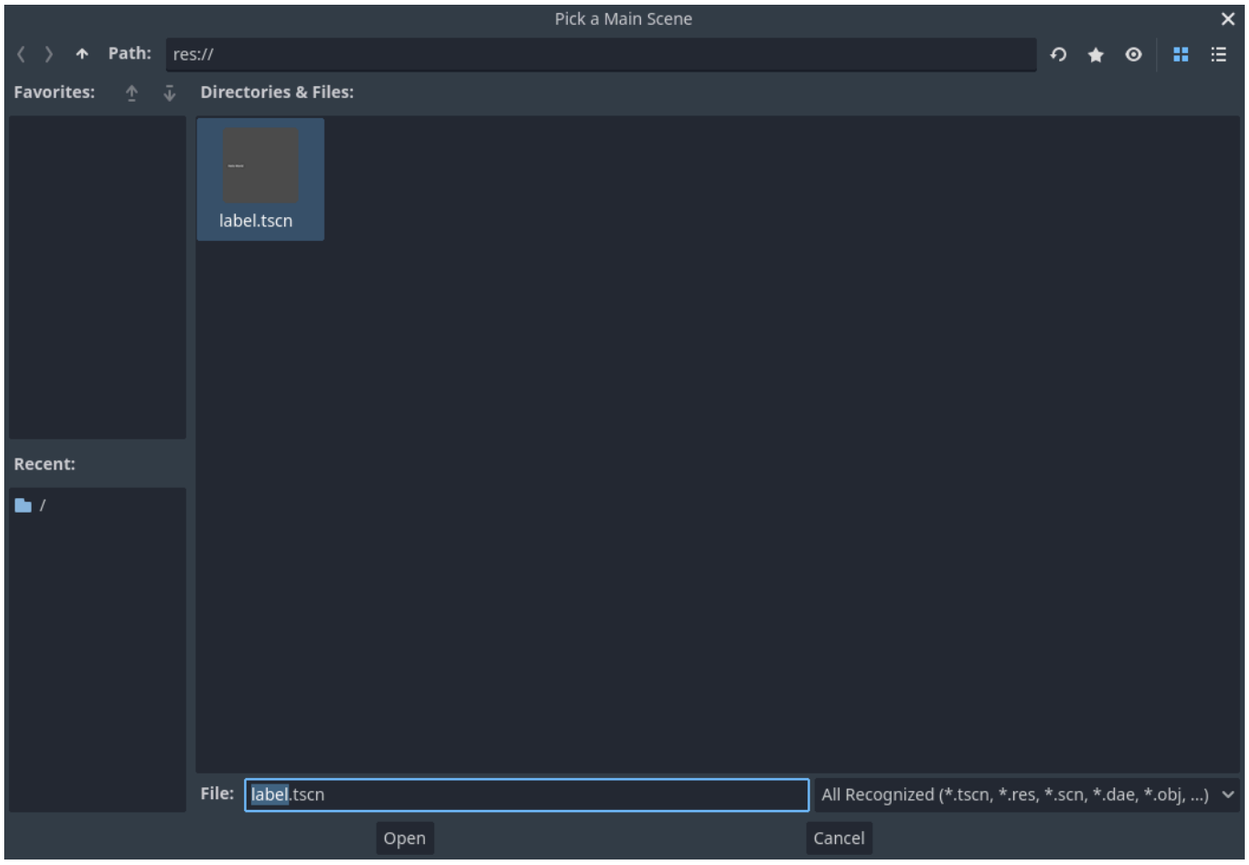
Для запуска тестовой сцены мы воспользовались кнопкой **Run Current Scene**. Еще одна кнопка рядом с ней позволяет установить и запустить основную сцену проекта. Для этого можно нажать F5



Появится всплывающее окно, предлагающее выбрать основную сцену.



Нажмите кнопку Select (Выбрать) и в появившемся диалоговом окне дважды щёлкните файл `label.tscn`.



Демонстрация должна запуститься снова. В дальнейшем, каждый раз, когда вы запускаете проект, Godot будет использовать эту сцену в качестве отправной точки.

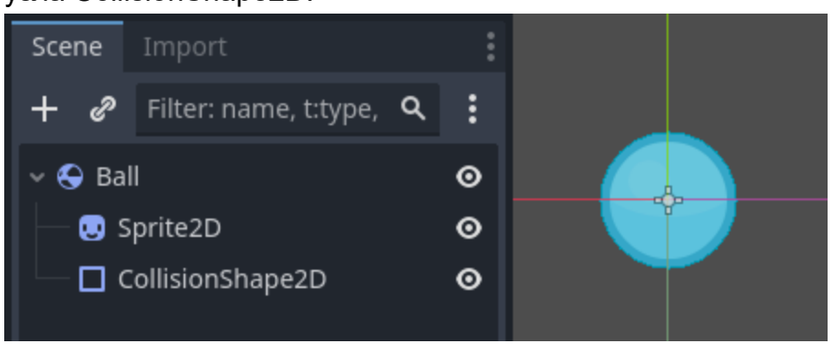
> Редактор сохраняет путь к главной сцене в файле **project.godot** в директории вашего проекта. Хотя вы можете редактировать этот текстовый файл напрямую, чтобы изменить настройки проекта, вы также можете использовать для этого окно "Project -> Project Settings"

Создание экземпляров

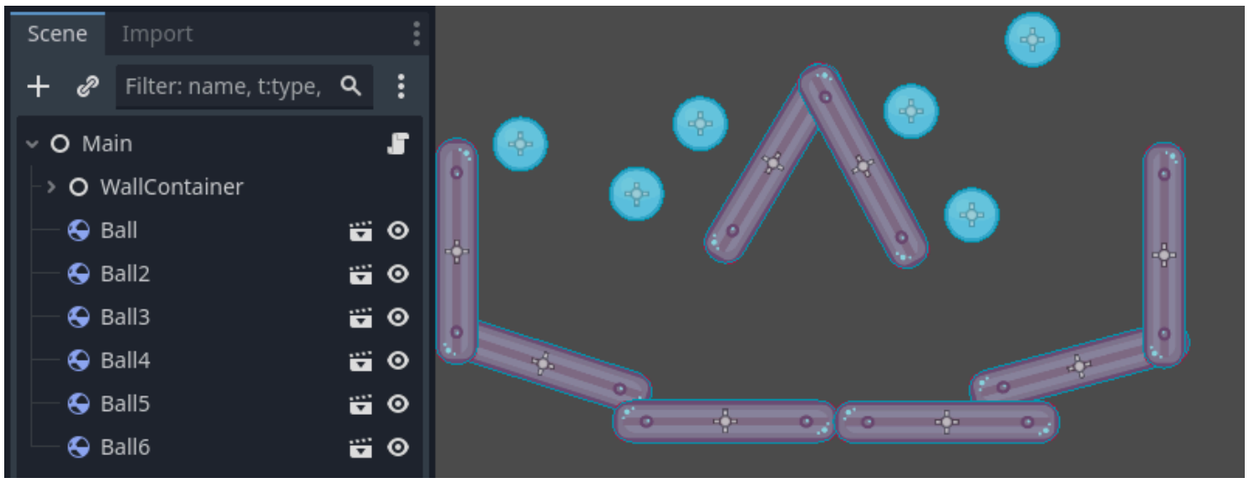
В предыдущей части мы увидели, что сцена - это набор узлов, организованных в древовидную структуру, корнем которой является один узел. Вы можете разделить свой проект на любое количество сцен. Эта функция поможет вам разбить и организовать различные компоненты вашей игры.

Вы можете создавать сколько угодно сцен и сохранять их как файлы с расширением `.tscn`, что означает "text scene". В качестве примера можно привести файл `label.tscn` из предыдущего урока. Мы называем эти файлы "упакованными сценами", поскольку они содержат информацию о содержимом сцены.

Вот пример мяча. Он состоит из узла RigidBody2D в качестве корня с именем Ball, который позволяет мячу падать и отскакивать от стен, узла Sprite2D и узла CollisionShape2D.



Если вы сохранили сцену, она работает как шаблон: вы можете воспроизводить её в других сценах столько раз, сколько захотите. Воспроизведение объекта по шаблону называется **инстансированием** (от слова "экземпляр").

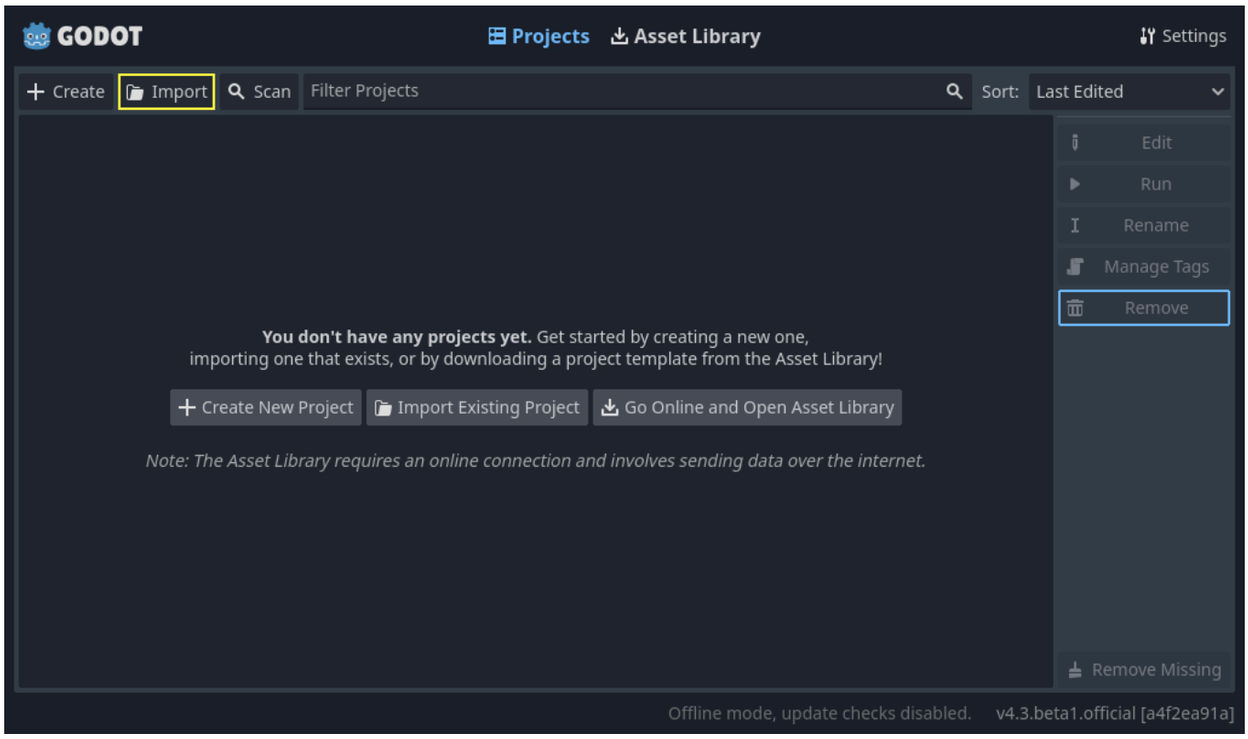


Как мы упоминали в предыдущей части, инстансированные сцены ведут себя как узел: редактор скрывает их содержимое по умолчанию. При экземпляре Ball отображается только узел Ball. Обратите внимание, что каждый дубликат имеет уникальное имя.

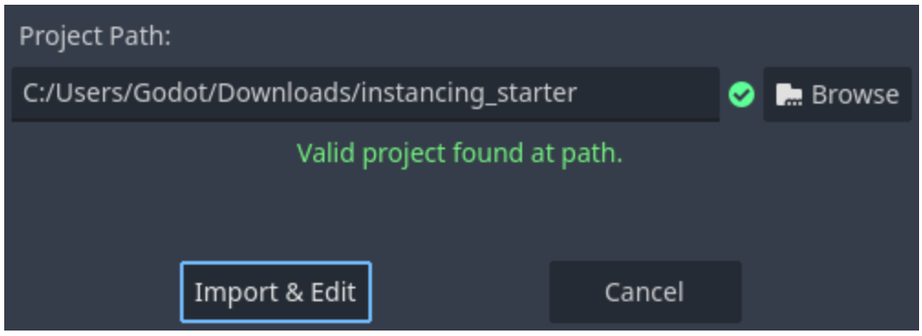
Каждый экземпляр сцены Ball начинает с той же структуры и свойств, что и `Ball.tscn`. Однако вы можете изменять каждый из них независимо, например, изменять то, как они отскакивают, сколько они весят, или любое свойство, открытое исходной сценой.

На практике

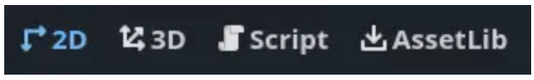
Давайте воспользуемся созданием экземпляров (инстансированием) на практике, чтобы увидеть, как именно оно работает в Godot.
Распакуйте архив на своём компьютере. Для его импорта вам потребуется Менеджер Проектов. Менеджер Проектов доступен при открытии Godot, либо, если вы уже открыли Godot, кликните на *Проект -> Выйти в список проектов*
В Менеджере Проектов нажмите кнопку _Import, чтобы импортировать проект.

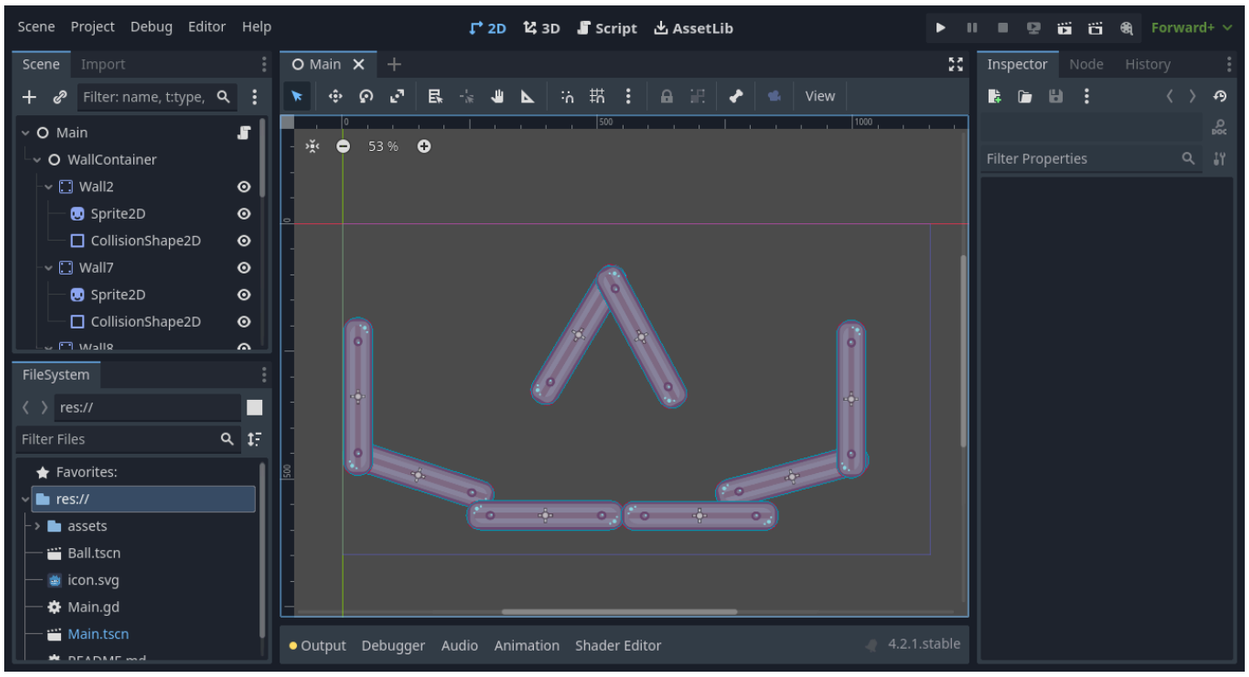


В появившемся окне перейдите в папку, которую вы извлекли. Дважды щёлкните по файлу `project.godot`, чтобы открыть его.
Наконец, нажмите кнопку **Import & Edit button** (Импорт и редактирование).

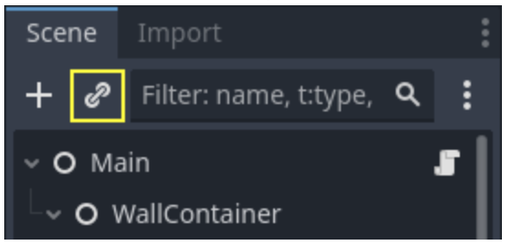


Проект содержит две упакованные сцены: `Main.tscn`, содержащую стены, о которые ударяется мяч, и `Ball.tscn`. Главная сцена должна открыться автоматически. Если вместо основной сцены вы видите пустую 3D сцену, кликните на кнопку 2D в верхней части экрана.

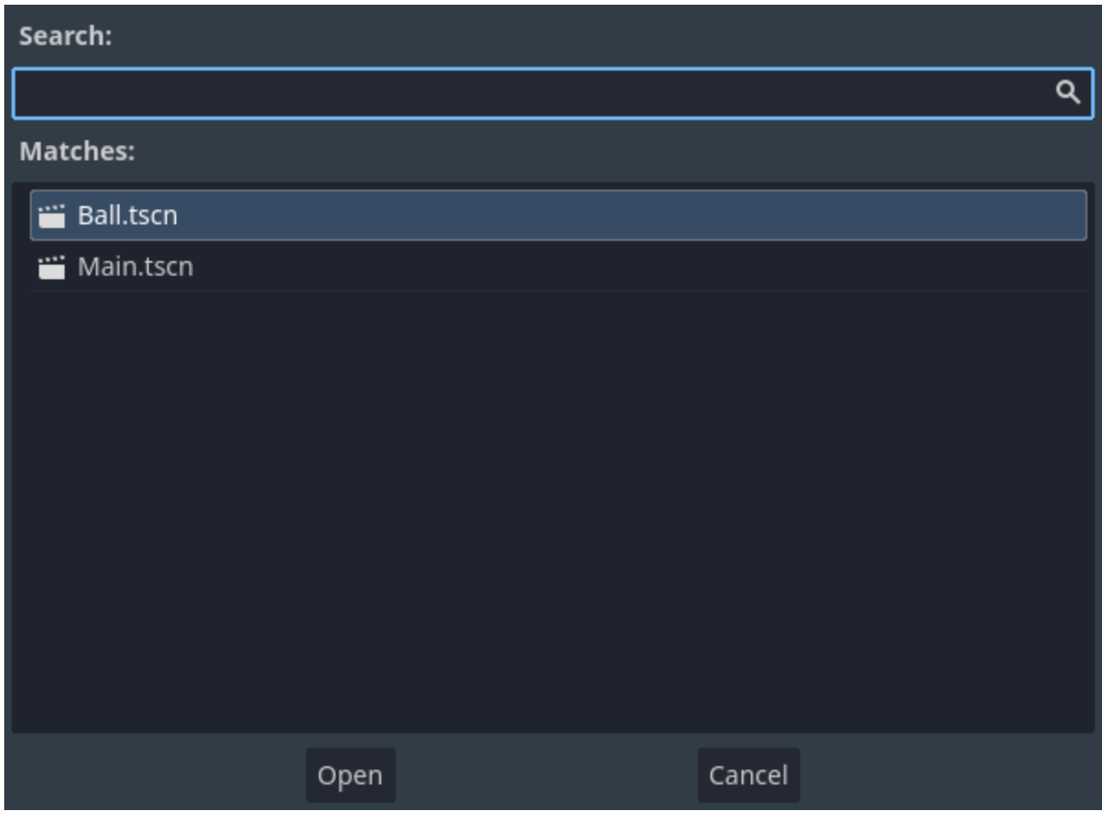




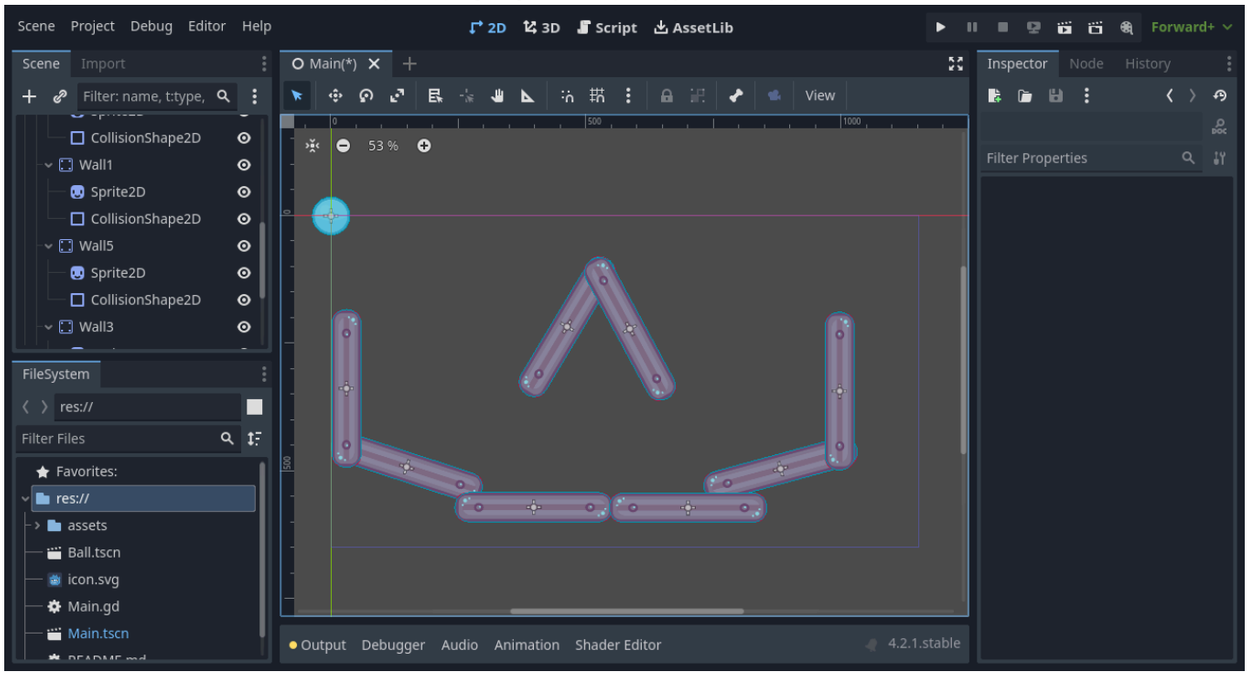
Добавим шар в качестве дочернего элемента узла Main. В панели Scene выберите узел Main. Затем щёлкните значок ссылки в верхней части панели Scene. Эта кнопка позволяет добавить экземпляр сцены в качестве дочернего узла выбранного в данный момент узла.



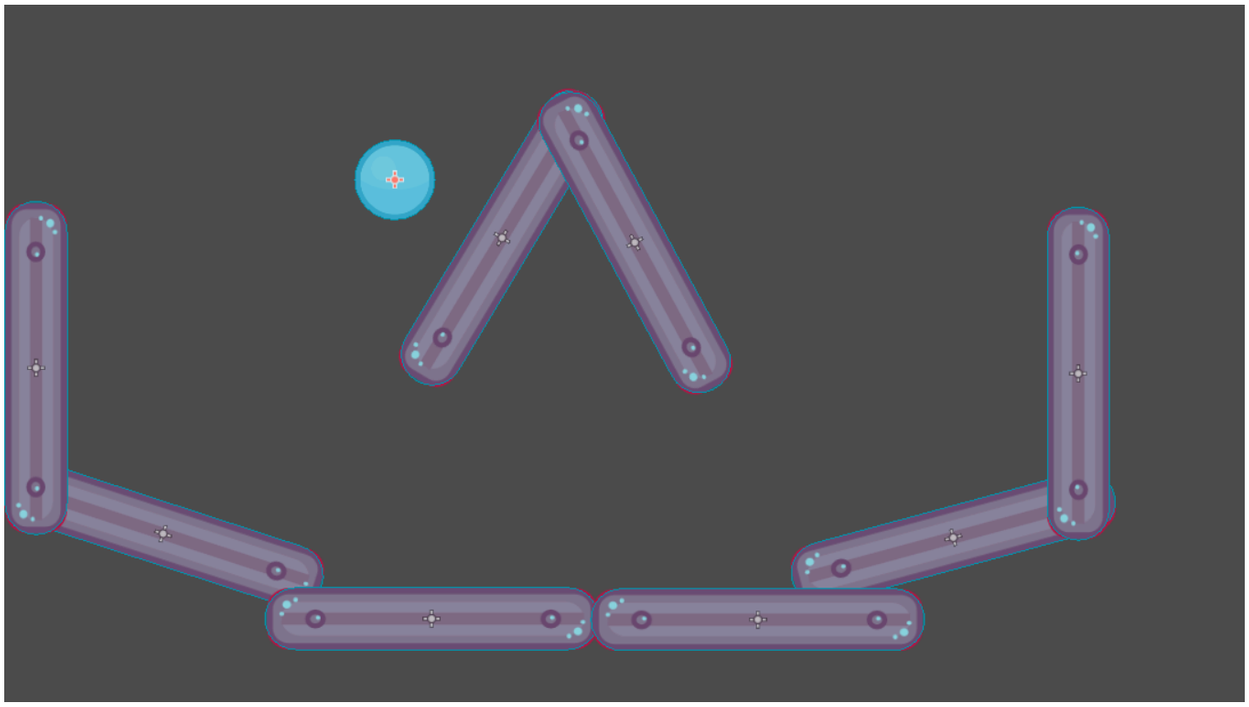
Дважды щелкните сцену с шаром, чтобы создать ее экземпляр.



Шар появляется в левом верхнем углу области просмотра.

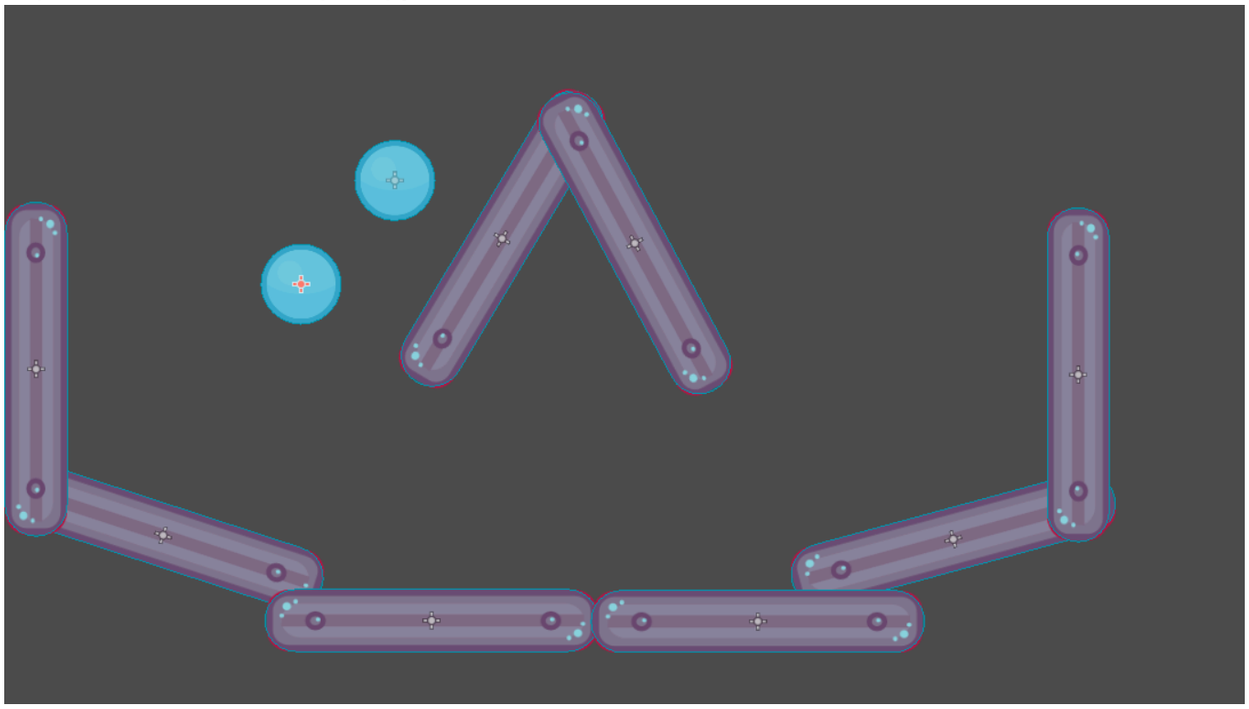


Нажмите на него и перетащите его к центру вида.

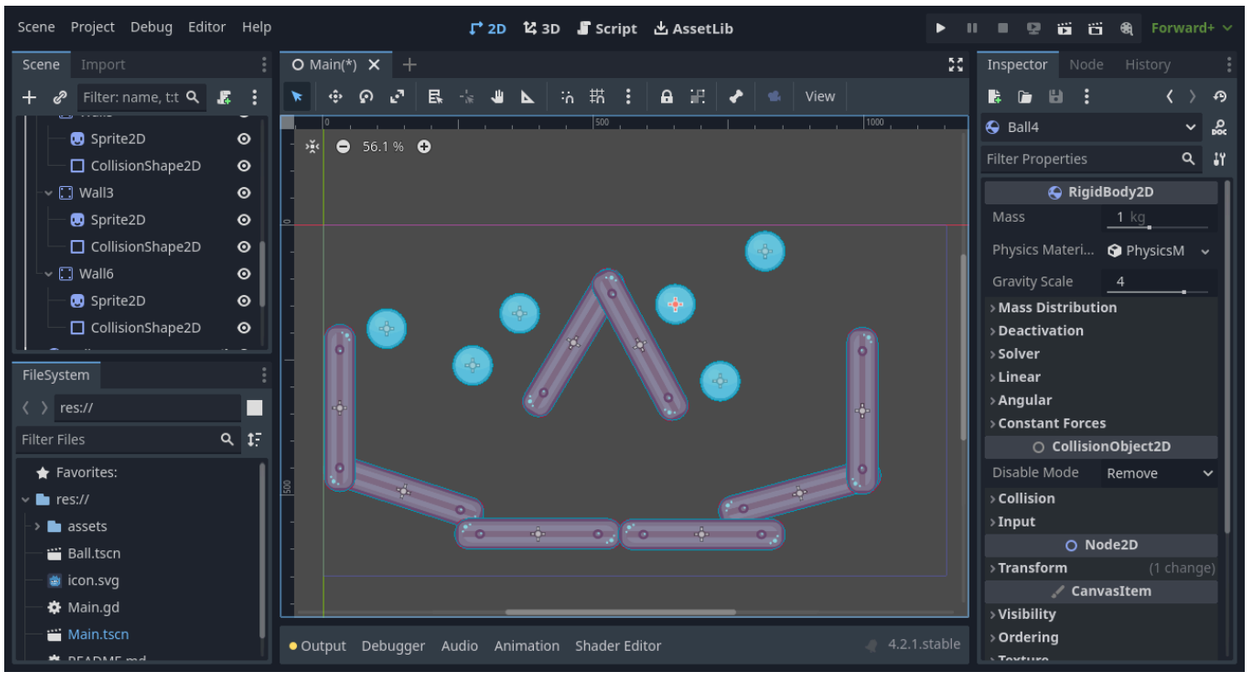


Запустите игру, нажав F5. Вы должны увидеть, как он падает.

Теперь мы хотим создать больше экземпляров узла Ball. Когда шар все ещё выделен, нажмите Ctrl-D, чтобы вызвать команду дублирования. Щелкните и перетащите, чтобы переместить новый шар в другое место.



Вы можете повторять этот процесс до тех пор, пока в сцене не появятся несколько.

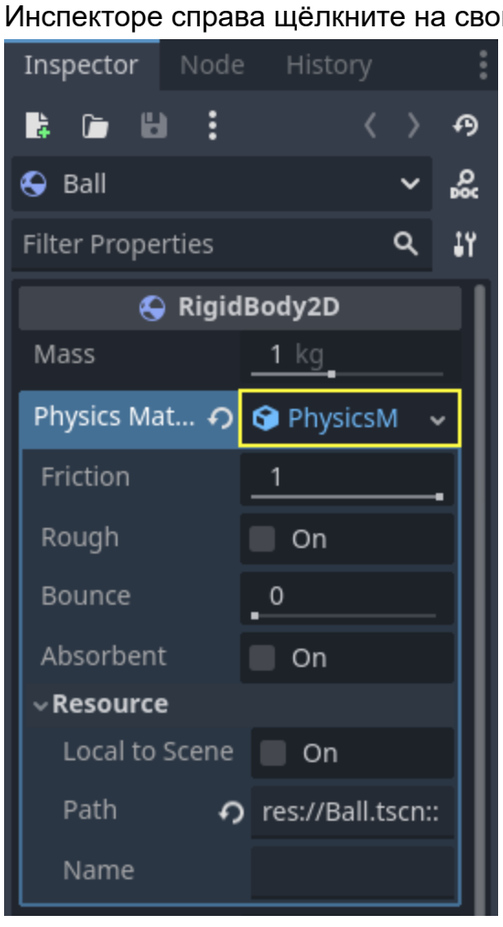


Запустите в игру ещё раз. Теперь вы должны увидеть, что каждый шарик падает независимо друг от друга. Это то, что делают экземпляры. Каждый из них является независимым воспроизведением шаблонной сцены.

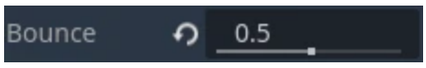
Редактирование сцен и экземпляров

И это ещё не всё. С помощью этой особенности вы можете:

1. Изменить свойства одного шара, не затрагивая другие, с помощью Инспектора.
2. Изменить свойства по умолчанию для каждого шара, открыв сцену `Ball.tscn` и внеся изменения в узел Ball там. После сохранения, все экземпляры Ball в проекте получат новые значения параметров.
   Давайте попробуем это сделать. Откройте файл `ball.tscn` и выберите узел Ball. В Инспекторе справа щёлкните на свойстве PhysicsMaterial, чтобы развернуть его.

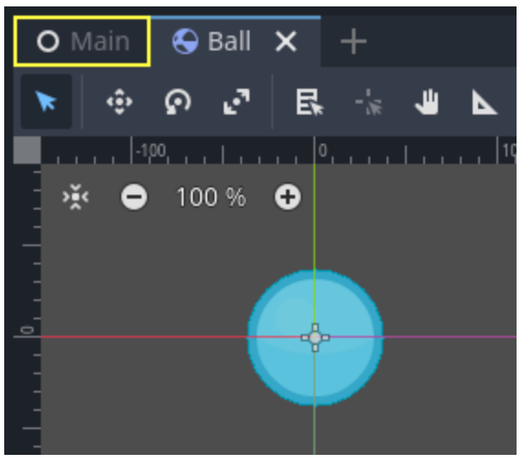


   Установите свойство Отскакивание (Bounce) в `0.5`, щелкнув по полю с цифрами, введя `0.5` и нажав Enter.

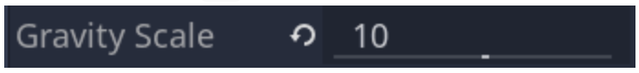


> Запустите игру, нажав F5, и обратите внимание, что все мячи теперь отскакивают гораздо сильнее. Поскольку сцена "Ball" является шаблоном для всех экземпляров, её изменение и сохранение приводит к соответствующему обновлению всех экземпляров.

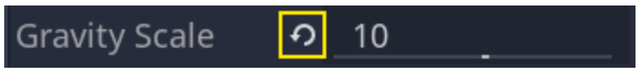
Теперь давайте настроим отдельный экземпляр. Вернитесь к главной сцене, нажав на соответствующую вкладку над окном просмотра.



Выберите один из экземпляров узла Ball и в инспекторе установите значение Gravity Scale равным `10`.



Рядом с измененным свойством появится серая кнопка «Вернуть».



Этот значок указывает на то, что вы переопределяете значение из исходной упакованной сцены. Даже если вы измените свойство в исходной сцене, переопределение значения будет сохранено в экземпляре. Нажатие на значок возврата приведет к восстановлению значения свойства в сохраненной сцене.

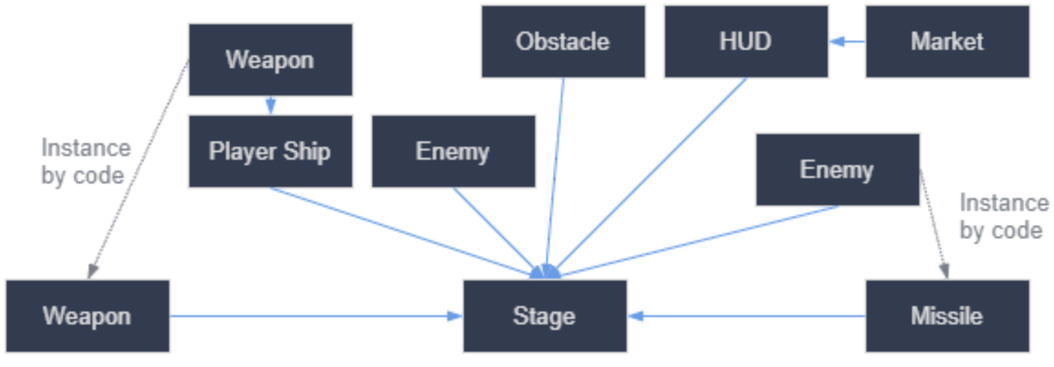
Запустите игру заново и обратите внимание, что теперь этот шарик падает гораздо быстрее остальных.

Экземпляры сцены как язык дизайна

Экземпляры и сцены в Godot предлагают отличный язык проектирования, выделяя движок среди других. Мы с самого начала разрабатывали Godot с учетом этой концепции.

При создании игр с Godot мы рекомендуем отказаться от архитектурных шаблонов кода, таких как Model-View-Controller (MVC) или диаграммы Entity-Relationship. Вместо этого вы можете начать с того, чтобы представить себе элементы, которые игроки увидят в вашей игре, и структурировать код вокруг них.

Например, вот как можно представить себе простой шутер:



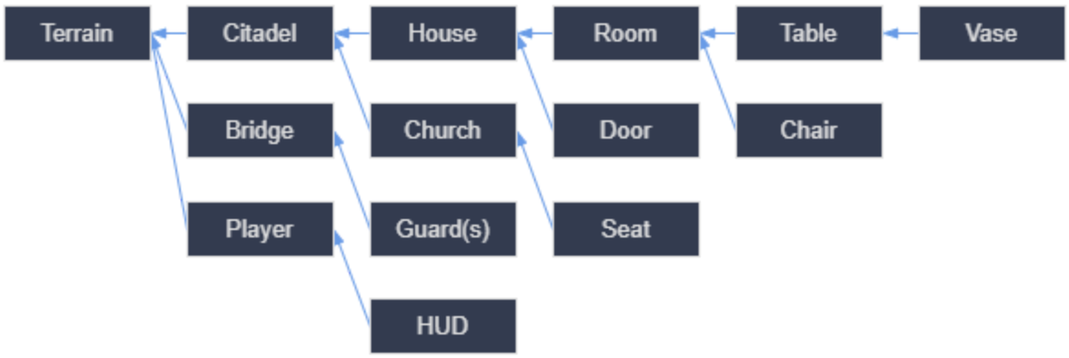
Подобную диаграмму можно придумать практически для любого типа игры. Каждый

прямоугольник представляет сущность, которая видна в игре с точки зрения игрока. Стрелки указывают, какая сцена чем владеет.

Когда у вас будет диаграмма, то для разработки вашей игры мы рекомендуем создать сцену для каждого элемента, перечисленного в ней. Для построения дерева сцен вы будете использовать инстансирование (создание экземпляров), либо с помощью кода, либо непосредственно в редакторе.

Программисты обычно тратят много времени на проектирование абстрактных архитектур и попытки вписать в них компоненты. Проектирование на основе сцен ускоряет и упрощает разработку, позволяя сосредоточиться на логике игры. Поскольку большинство компонентов игры напрямую связаны со сценами, использование дизайна, основанного на экземплярах сцен, означает, что вам не нужно много другого архитектурного кода.

Вот пример диаграммы сцены для игры с открытым миром с множеством ресурсов и вложенных элементов:



Представьте, что мы начали с создания комнаты. Мы могли бы сделать пару разных комнатных сцен с уникальной расстановкой мебели в них. Позже мы могли бы создать сцену дома, в которой для интерьера использовалось бы несколько экземпляров комнаты. Мы создадим цитадель из множества экземпляров домов и большого ландшафта, на котором мы разместим цитадель. Каждая из них будет сценой, создающей одну или несколько вложенных сцен.

Позже мы можем создать сцены, которые представляют собой стражников, и добавить их в нашу крепость. В результате, они будут косвенно добавлены в общий игровой мир.

С помощью Godot можно легко итерационно дорабатывать игру, ведь всё, что вам нужно - это создавать и инстансировать новые сцены. Мы разработали редактор так, чтобы он был доступен и программистам, и дизайнерам, и художникам. В типичном процессе командной разработки могут участвовать 2D или 3D художники, дизайнеры уровней, гейм-дизайнеры и аниматоры, и все они работают с редактором Godot.

Подведение итогов

Инстансирование, процесс создания объекта из шаблона имеет множество полезных применений. Со сценами это даёт вам:

- Возможность разделить свою игру на многократно используемые компоненты.
- Инструмент для структурирования и инкапсуляции сложных систем.
- Язык, позволяющий думать о структуре вашего игрового проекта естественным образом.

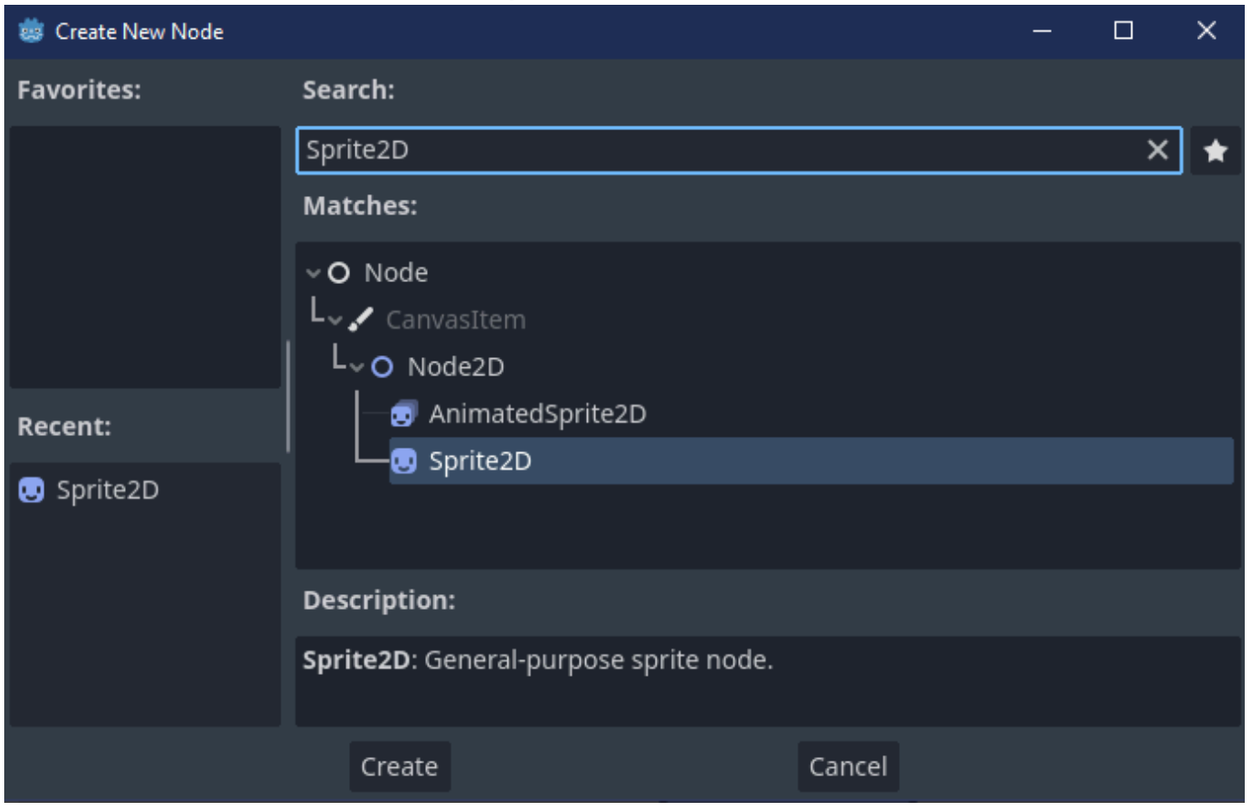
Создание вашего первого скрипта

Настройка проекта

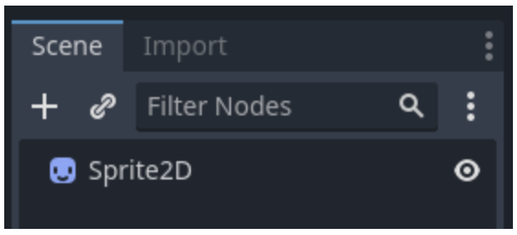
Пожалуйста, создайте новый проект, чтобы начать с чистого листа. Ваш проект должен содержать одно изображение: иконку Godot, которую мы в сообществе часто используем для прототипирования.

Нам нужно создать узел Sprite2D для отображения в игре. В панели Сцена (Scene), кликните на кнопку Другой Узел (Other Node).

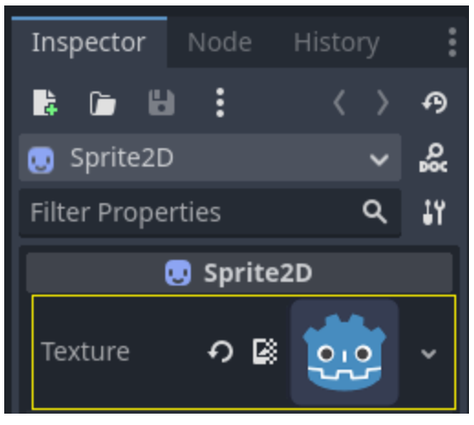
Наберите "Sprite2D" в поисковой строке, чтобы отобразились нужные узлы, затем дважды кликните на Sprite2D, чтобы создать узел.



Вкладка Сцена (Scene) теперь должна содержать только узел Sprite2D.

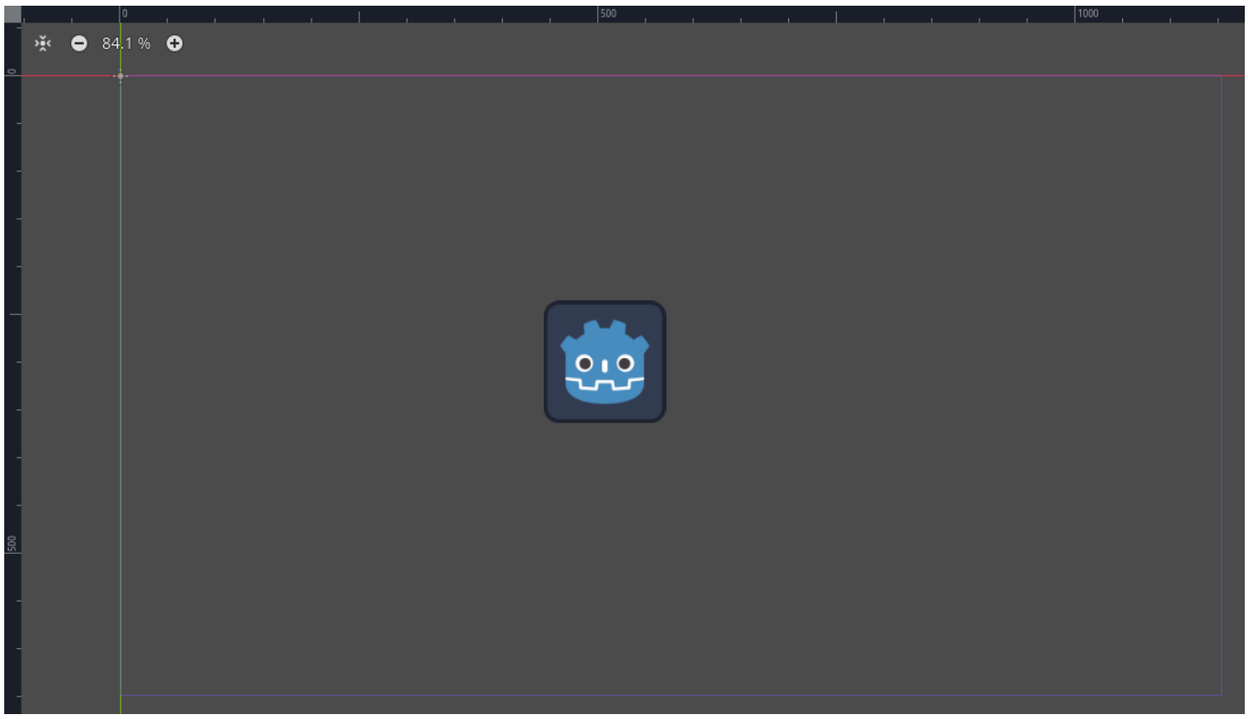


Узлу Sprite2D нужна текстура для отображения. В Инспекторе справа вы можете увидеть, что в свойстве текстуры указано "[пусто]". Чтобы отобразить иноку Godot, кликните и перетащите файл `icon.png` из файловой системы панели на слот текстуры.



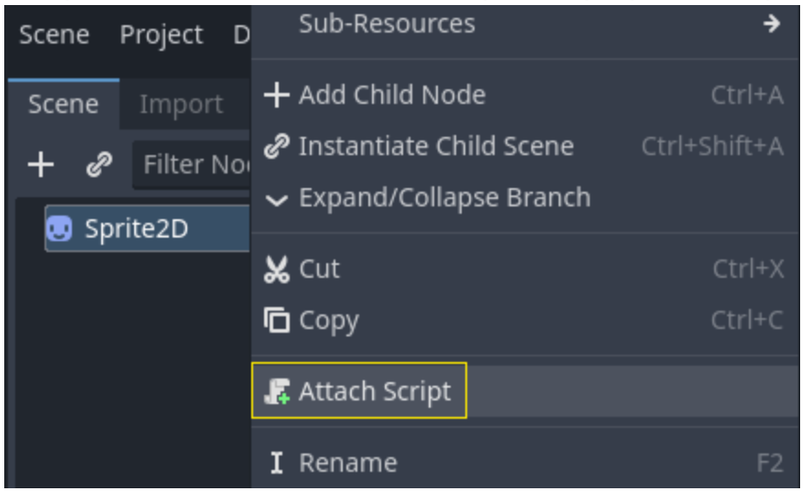
> Также вы можете создавать узлы Sprite2D автоматически, перетаскивая изображения в Окно просмотра.

Затем нажмите и перетащите значок в окне просмотра, чтобы отцентрировать его в игровом представлении.



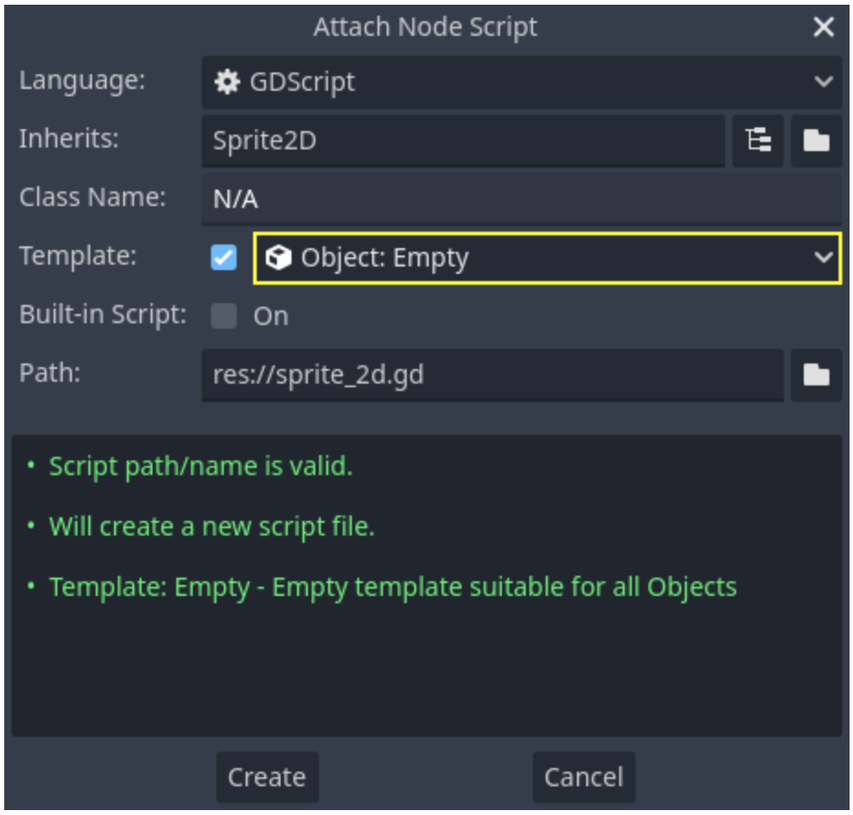
Создание нового скрипта

Чтобы создать и прикрепить новый скрипт к нашему узлу, нажмите правой кнопкой мыши на Sprite2D в панели сцены и выберите "Прикрепить Скрипт" (Attach Script).



Откроется окно "Прикрепить скрипт" (Attach Node Script). В этом окне вы можете указать язык программирования скрипта, путь в корневой папке и другие параметры.

Измените Template (Шаблон) с Default на Empty, чтобы начать с чистого файла. Оставьте остальные параметры по умолчанию и нажмите кнопку Create (Создать), чтобы создать скрипт.



В рабочей области Script должен появиться новый файл `sprite_2d.gd` и следующая строка кода:

```
extends Sprite2D
```

Каждый файл GDScript неявно является классом. Ключевое слово `extends` определяет класс, который наследует или расширяет данный скрипт. В этом случае `Sprite2D` означает, что наш скрипт получит доступ к свойствам и функциям узла `Sprite2D`, включая классы, которые он расширяет, такие как `Node2D`, `CanvasItem` и `Node`.

> В GDScript, если вы опустите строку с ключевым словом `extends`, ваш класс будет неявно расширять RefCounted, который Godot использует для управления памятью вашего приложения.

К унаследованным свойствам относятся те, которые вы можете видеть в инспекторе, например, `texture` узла.

> По умолчанию, Инспектор отображает свойства узла в "Title Case", с заглавными буквами вначале, разделенные пробелом. В GDScript эти свойства записываются в "snake_case", строчными буквами, разделенными подчёркиванием.

Привет, мир!

Наш скрипт ничего не делает. Давайте заставим его вывести "Hello, world!" в панель вывода в нижней части экрана.

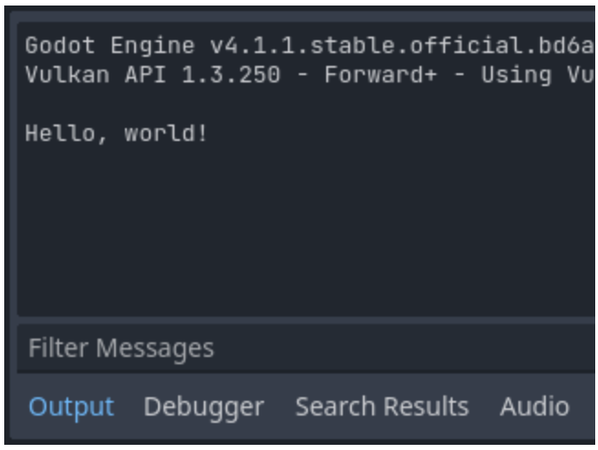
Добавьте следующий код в ваш скрипт:

```
func _init():
        print("Hello, world!")
```

Давайте разберём это. Слово `func` объявляет новую функцию `_init`. Это конструктор нашего класса. Движок вызывает `_init` при создании объекта в памяти, если вы определили эту функцию.

> GDScript - язык с отступами. Отступ (табуляция) в начале строки `print()` обязателен для работы кода. Если вы пропустите отступ или отступите неправильно, редактор выделит строку красным и выведет сообщение:"Indented block expected" ("Блок отступа пропущен").

Сохраните сцену как `sprite_2d.tscn`, если ещё не сделали этого, затем нажмите F6 (Cmd + R на macOS) для запуска. Посмотрите в окно вывода **Вывод** (Output) в панели внизу. Там должно отобразится "Hello, world!".



Удалите функцию `_init()`, оставив только строку `extends Sprite2D`.

Поворот вокруг

Теперь сделаем наш узел двигающимся и вращающимся. Для этого добавим две переменных-члена в наш скрипт: скорость перемещения в пикселях в секунду и угловую скорость в радианах в секунду. Добавим следующий код после строки `extends Sprite2D`.

```
var speed = 400
var angular_speed = PI
```

Переменные-члены располагаются в начале скрипта, после всех строк "extends", но перед функциями. Каждый экземпляр узла с прикреплённым к нему скриптом имеет собственную копию свойств `speed` и `angular_speed`.

> Углы в Godot по умолчанию задаются в радианах, как и в некоторых других движках, но вы можете использовать встроенные функции и свойства для работы с градусами.

Для перемещения иконки мы должны обновлять её позицию и вращение каждый кадр игрового цикла. Для этого используем виртуальную функцию `_process()` класса `Node`. При объявлении этой функции в любом классе, наследуемом от `Node` (например, `Sprite`), Godot будет вызывать её каждый кадр и передавать в аргументе `delta` время, прошедшее от предыдущего кадра.

> Игры работают в цикле, отображая множество изображений в секунду, каждое из которых называется кадром. Скорость, с которой игра создаёт эти изображения, измеряется в кадрах в секунду (Frame Per Second). Большинство игр нацелены на 60 FPS, хотя вы можете найти такие цифры, как 30 кадров в секунду на более медленных мобильных устройствах или от 90 до 240 для игр виртуальной реальности.
> Движок и разработчики игры делают все возможное, чтобы обновлять игровой мир и рендерить изображения с постоянным интервалом времени, но всегда существуют небольшие отклонения во времени рендеринга кадров. Поэтому движок предоставляет нам это значение дельта-времени, делая наше движение независимым от частоты кадров.

В нижней части скрипта определите функцию:

```
func _process(delta):
        rotation += angular_speed * delta
```

Ключевое слово `func` определяет (создаёт) новую функцию. После него в нужно написать имя функции и в скобках аргументы, которые она принимает. Двоеточие завершает определение, а следующие за ним блоки с отступом представляют собой содержимое или инструкции функции.

> Обратите внимание, что `_process()`, как и `_init()`, начинается с символа подчёркивания. По соглашению, виртуальные функции, то есть встроенные функции Godot, которые вы можете переопределить - начинаются с символа подчёркивания.

Строка внутри функции, `rotation += angular_speed * delta`, увеличивает угол поворота нашего спрайта каждый кадр. Здесь `rotation` — это свойство, унаследованное от класса `Node2D`, который расширяет класс `Sprite2D`. Он контролирует поворот нашего узла и работает с радианами.
Запустите сцену, чтобы увидеть, что иконка Godot крутится на месте.

Движение вперёд

Давайте теперь заставим узел двигаться. Добавьте следующие две строки внутри функции `_process()`, убедившись, что новые строки имеют такой же отступ, как и строка `rotation += angular_speed * delta` перед ними.

```
var velocity = Vector2.UP.rotated(rotation) * speed

position += velocity * delta
```

Как мы уже видели, ключевое слово `var` определяет новую переменную. Если вы поместите его в верхней части скрипта, оно определит свойство класса. Внутри функции оно определяет локальную переменную: данная переменная существует только в области действия функции.

Мы определяем локальную переменную `velocity`, 2D-вектор, представляющий собой направление и скорость. Чтобы заставить узел двигаться вперёд, мы берём константу `Vector2.UP` класса Vector2, вектор, указывающий вверх, и вращаем его, вызывая метод `rotated()`. Данное выражение `Vector2.UP.rotated(rotation)` - вектор, указывающий вперёд относительно нашей иконки. Умноженное на наше свойство `speed`, это выражение даёт нам скорость, которую мы можем использовать для перемещения узла вперёд.

Мы добавляем `velocity * delta` к узлу `position`, чтобы переместить его. Сама позиция имеет тип Vector2, встроенный тип в Godot, представляющий двумерный вектор.

Запустите сцену, чтобы увидеть, что голова Godot движется по кругу.

Готовый скрипт

```
extends Sprite2D

var speed = 400
var angular_speed = PI


func _process(delta):
```

```
        rotation += angular_speed * delta

        var velocity = Vector2.UP.rotated(rotation) * speed

        position += velocity * delta
```

Отслеживание ввода игрока

Основываясь на предыдущем уроке Создание вашего первого скрипта, давайте рассмотрим еще одну важную особенность любой игры: предоставление контроля игроку. Чтобы добавить это, нам нужно изменить наш код `sprite_2d.gd`.

У вас есть два важных инструмента для обработки пользовательского ввода в Godot:

1. Встроенные обратные вызовы ввода, в основном `_unhandled_input()`. Подобно функции `_process()` функция `_unhandled_input()` — это встроенная виртуальная функция, которую Godot вызывает каждый раз, когда игрок нажимает клавишу. Это инструмент, который вы захотите использовать, чтобы реагировать на события, которые не происходят каждый кадр, например, нажатие Space для прыжка.
2. Синглтон `Input`. *Синглтон* - это глобально доступный объект. Godot предоставляет доступ к нескольким из них в скриптах. Это подходящий инструмент для проверки ввода в каждом кадре.

Здесь мы будем использовать синглтон `Input`, поскольку нам нужно знать в каждом кадре, хочет ли игрок поворачиваться или двигаться.

Для поворота нам стоит использовать новую переменную: `direction`. В нашей функции `_process()` замените строку `rotation += angular_speed * delta` на приведенный ниже код.

```
var direction = 0
if Input.is_action_pressed("ui_left"):
        direction = -1
if Input.is_action_pressed("ui_right"):
        direction = 1

rotation += angular_speed * direction * delta
```

Наша локальная переменная `direction` - это множитель, представляющий собой направление, в котором игрок хочет повернуть. Значение `0` означает, что игрок не нажимает ни стрелку влево, ни стрелку вправо. Значение `1` говорит, что игрок хочет повернуть вправо, а `-1` - что игрок хочет повернуть влево.

Для обработки этих значений мы вводим условие и используем `Input`. Условие в GDScript начинается с ключевого слова `if` и заканчивается двоеточием. Условие - это выражение между ключевым словом и концом строки.

Чтобы проверить, была ли нажата клавиша в этом кадре, мы вызываем `Input.is_action_pressed()`. Метод принимает текстовую строку, представляющую входное действие, и возвращает `true`, если клавиша нажата, и `false` в противном случае.

Два действия, которые мы использовали выше, «ui_left» и «ui_right», предопределены в каждом проекте Godot. Они соответственно срабатывают, когда игрок нажимает стрелки влево и вправо на клавиатуре или же влево и вправо на крестовине геймпада.

> Вы можете просматривать и редактировать действия ввода в своем проекте, перейдя в «Проект» -> «Настройки проекта» и щелкнув вкладку «Список действий».

Наконец, мы используем `direction` как множитель, когда обновляем `rotation` узла: `rotation += angular_speed * direction * delta`.

Закомментируйте строки `var velocity = Vector2.UP.rotated(rotation) * speed` и `position += velocity * delta` следующим образом:

```
#var velocity = Vector2.UP.rotated(rotation) * speed

#position += velocity * delta
```

Это игнорирует код, который перемещал иконку по кругу без ввода пользователя из предыдущего упражнения.

Если запустить сцену с этим кодом, иконка должна вращаться при нажатии Влево и Вправо.

Перемещение при нажатии "вверх"

Чтобы двигаться только при нажатии клавиши, нам нужно изменить код, который вычисляет скорость. Раскомментируйте код и замените строку, начинающуюся с `var velocity`, на код ниже.

```
var velocity = Vector2.ZERO
if Input.is_action_pressed("ui_up"):
        velocity = Vector2.UP.rotated(rotation) * speed
```

Мы инициализируем `velocity` со значением `Vector2.ZERO`, еще одной константой встроенного типа `Vector`, представляющей двумерный вектор нулевой длины.

Когда игрок выполняет действие "ui_up", мы обновляем значение скорости, заставляя спрайт двигаться вперёд.

Готовый скрипт

```
extends Sprite2D

var speed = 400
var angular_speed = PI


func _process(delta):
        var direction = 0
        if Input.is_action_pressed("ui_left"):
                direction = -1
        if Input.is_action_pressed("ui_right"):
                direction = 1

        rotation += angular_speed * direction * delta

        var velocity = Vector2.ZERO
        if Input.is_action_pressed("ui_up"):
                velocity = Vector2.UP.rotated(rotation) * speed
```

```
        position += velocity * delta
```

Подведение итогов

В общем, каждый скрипт в Godot представляет собой класс и расширяет один из встроенных классов движка. Типы узлов, от которых наследуются ваши классы, дают вам доступ к таким свойствам, как `rotation` и `position` в случае нашего спрайта. Вы также наследуете множество функций, которые мы не использовали в этом примере.

В GDScript переменные, которые вы помещаете в верхней части файла, являются свойствами вашего класса, также называемыми переменными-членами. Кроме переменных, вы можете определять функции, которые, по сути, являются методами ваших классов.

Godot предоставляет несколько виртуальных функций, которые вы можете определить для связи вашего класса с движком. К ним относятся `_process()`, применяющая изменения к узлу каждый кадр, и `_unhandled_input()`, получающая события ввода, такие как нажатие клавиш и кнопок от пользователей. Есть еще множество других функций.

Синглтон `Input` позволяет Вам реагировать на ввод от игрока в любом месте Вашего кода. В первую очередь Вы сможете применить его в цикле `_process()`.